Michael Dahl

Question 1:
According to goldgatebridge.org, the width of the Golden Gate bridge is 90 ft and the length including approaches from abutment to abutment is 8981 feet. Multiply the length times the width to get the area of the bridge:

8981 * 90 = 808,290 sq ft.

My own measurement for the diameter of a penny is 3/4 of an inch. Because of their circular shape, pennies laid adjacent to one another will actually take up the same amount of area as a square with .75 inch length sides. To get the area, multiply the diameter times the diameter:

.75 * .75 = .5625 sq in.

Finally, divide the area of the bridge by the area a penny occupies to find the number of pennies that will fit on the bridge. But first make the units of the areas the same:

808,290 sq ft. * 12 = 9,699,480 sq in.

9,699,480 sq in. / .5625 sq in. = 17,243,520 pennies

Question 2:
Assume that the design layout has 12 pixel increments with no margins and there are no limits on the image height. In order to fit into the design layout, a 16 x 9 image must have a width that is a member of the sequence 12, 24, 36, 48,…

Three images sizes that satisfy this condition are: 48 x 27
96 x 54
144 x 81

Question 3:
The minimum number of moves for any chess piece to cover the chessboard would have to be 64 since the board contains 64 squares. The question is whether or not a knight with its' interesting L shaped move can cover the entire board while touching each of the squares only once. It turns out that this puzzle is known as the Knight's Tour problem and can be solved using either brute force or a heuristic. Warnsdorff's algorithm is a heuristic method for solving the Knight's Tour, based on the rule of choosing the square among those immediately accessible by the knight move that would give the fewest possible knight's moves following the move to that square.

The Java program displayed below uses Warnsdorff's algorithm to show that it is possible for a knight to cover the chessboard in 64 moves.

Code:

```
public class ChessBoard
{
       private int moveNumber;
       private int[][] board;
       private int boardLength;
       private int boardWidth;
       private int numberOfMoves;

       int horizontal[] = {2, 1, -1, -2, -2, -1, 1, 2};
       int vertical[] = {-1, -2, -2, -1, 1, 2, 2, 1};

        public int[] initialization()
         {
             moveNumber = 0;
             numberOfMoves = 8;
             boardLength = 8;
             boardWidth = 8;

             int[] square = new int[2];
             square[0] = (int)Math.random() * boardLength;
             square[1] = (int)Math.random() * boardWidth;
             board = new int[boardLength][boardWidth];
             board[square[0]][square[1]] = ++moveNumber;

             return square;
         }

         public int[] nextMove(int[] square)
         {
            int access = numberOfMoves;
            int row = square[0];
            int column = square[1];


            for(int i = 0; i < numberOfMoves; i++)
            {
                int tempRow = row + vertical[i];
                  int tempColumn = column + horizontal[i];

               int tempAccess = getAccessibility(tempRow, tempColumn);

               if(validMove(tempRow, tempColumn) && tempAccess < access )
               {
                  square[0] = tempRow;
                  square[1] = tempColumn;
                  access = tempAccess;
               }
            }

            board[square[0] ][square[1] ] = ++moveNumber;

            return square;
         }
```

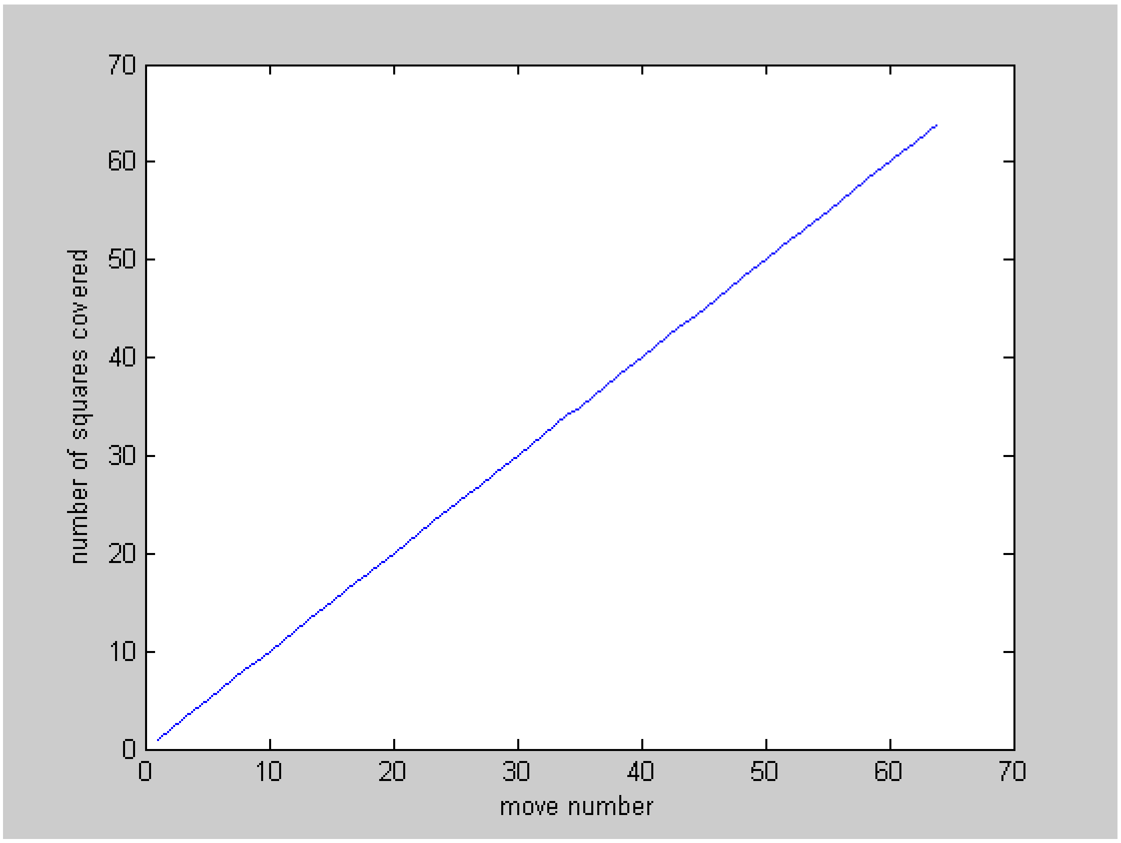
```
    private int getAccessibility(int row, int column)
    {
       int access = 0;
       for(int i = 0; i < numberOfMoves; i++)
       {
          if(validMove(row + vertical[i], column + horizontal[i]))
             access++;
       }
       return access;
    }

    private boolean validMove(int row, int column)
    {
       return ( row < boardLength  && row >= 0 && column < boardWidth
                 && column >=0  &&
           board[row][column] == 0 );
    }

    public void printBoard()
    {
       for (int i=0; i < boardLength ; i++)
       {
          for (int j = 0; j < boardWidth; j++)
          {
             System.out.print(board[i][j] + "\t");
          }
          System.out.print("\n");
       }
       System.out.print("\n");
       System.out.print("The number of moves was: " + moveNumber);
    }

    public int getSize()
    {
        return boardLength * boardWidth;
    }



  public static void main(String[] args)
  {
      ChessBoard cb = new ChessBoard();
      int[] position = cb.initialization();

      for (int i=1; i< cb.getSize(); i++)
      {
          position = cb.nextMove(position);
      }

      cb.printBoard();

  }
}
```

Output:

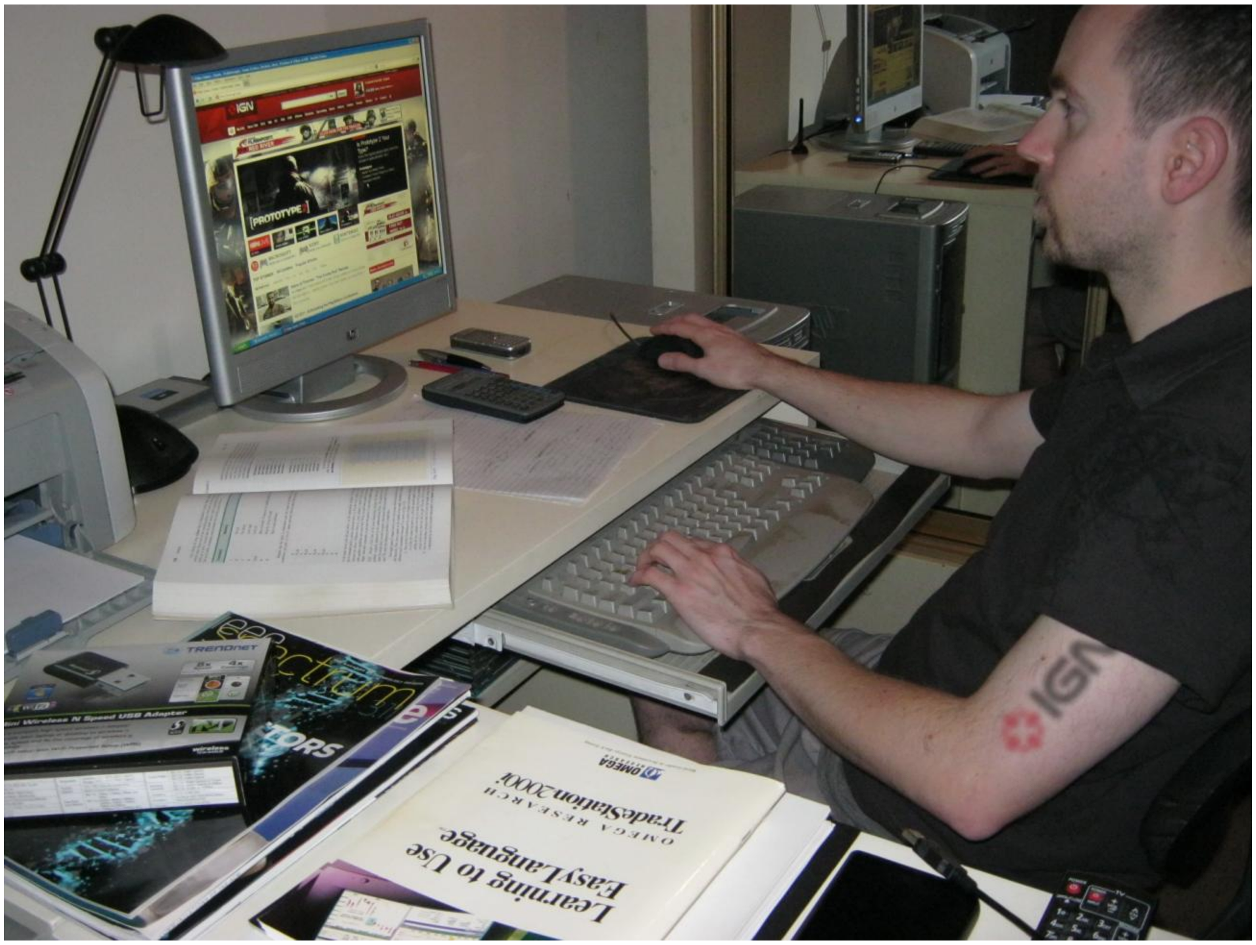
```
1       22      3       18      25      30      13      16
4       19      24      29      14      17      34      31
23      2       21      26      35      32      15      12
20      5       56      49      28      41      36      33
57      50      27      42      61      54      11      40
6       43      60      55      48      39      64      37
51      58      45      8       53      62      47      10
44      7       52      59      46      9       38      63

The number of moves was: 64
```

The graph of the move number vs. squares covered is linear with one additional square being covered on each move.

MATLAB plot:



Question 4:


IGN
[PROTOTYPE]
TRENDnet
Learning to Use EasyLanguage
OMEGA RESEARCH TradeStation 2000i
OMEGA
IGN